**Linux Networking**

**CDN Service for Edge Cloud VPC Environment**

**Project Milestone 3**

**Presented by:**

**Sandeep Kumar Balaga 200312527**

**Saurabh Patil 200318193**

**John Emmanuel Palagiri 200270155**

**Jayesh Chandrashekar Jakkani 200321561**

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Content Title** | | **Page No.** |
| 1. | **Introduction** | | **3** |
|  | **1.1** | **Background** | **3** |
|  | **1.2** | **Problem Description** | **3** |
|  | **1.3** | **Detailed Project Goal** | **3** |
| **2.** | **Related Work** | | **4** |
|  | **2.1** | **CDN** | **4** |
|  | **2.2** | **VPC** | **5** |
|  | **2.3** | **DNS** | **6** |
| **3.** | **Project Description** | | **7** |
|  | **3.1** | **System Description** | **7** |
|  | **3.2** | **Proposed Features** | **7** |
|  | **3.3** | **Topology Description** | **9** |
| **4.** | **Implementation Architecture** | | **11** |
|  | **4.1** | **Implementation Details** | **11** |
|  | **4.2** | **Tenant Guide** | **12** |
|  | **4.3** | **Developer Guide** | **12** |
| **5.** | **Summary** | | **13** |
| **6.** | **Future Scope** | | **13** |

**1. Introduction**

**1.1 Background:** The aim of the project is to deploy a virtual private cloud for a tenant who wants to provide content service to its customers. This project allows the tenant to create its own infrastructure without setting up the connection, maintaining desired IP address ranges, configuring servers, some fault management and analytical features by using open source software. This set of features will give tenants an open hand on scaling their infrastructure depending on their business requirement.

**1.2 Problem Description:** The main problem for tenants will be accessing its virtual private cloud. To solve this issue, a private DNS server is deployed within the VPC. Along with tenants, users will also have difficulties accessing the tenant’s content. This can be eliminated by configuring DNS server to solve IP queries and Nginx web server to solve content queries. Another issue that we have to take care of is edge breakdown or edge mobility as this service should be available to both tenants and users at all times.

**1.3 Detailed Project Goal:** The main goal of this project is to create Virtual Private Cloud for a tenant so that it can host its web content on edge device which will be accessed by localized users. This service is automated to minimize the time for deploying and managing the infrastructure which is achieved by using open source tools and software. To make this service up and running, we need to implement two servers i.e. DNS and Nginx. DNS server will maintain connectivity between the server in different clouds and Nginx web server will maintain the logs of content within the edge device. Infrastructure is created by taking inputs from the tenant using json script and automation is done using ansible so as to maintain the speed for creation. VM and containers are used to build this infrastructure. First, we used VM to achieve the goal but after knowing the benefits of docker container i.e. builds infrastructure in less time, we shifted towards a better option . The system is developed using open source utilities like ansible for infrastructure automation, Linux facilities such as iptables, namespaces, etc., nginx and BIND9 to design DNS server.

**2. Related Work**

* 1. **CDN:**

2.1.1 Amazon CloudFront:

Amazon CloudFront is a web service that speeds up distribution of tenant’s static and dynamic web content, such as .html, .css, .js, and image files, to tenant’s users. CloudFront delivers tenant’s web content by using network of data centers called edge locations. When the user requests for a tenant’s content, CloudFront will deliver this data through a edge location which will provide lowest latency and thus serving the content through the best possible route. On the other hand, If the content is not present in that edge location then CloudFront retrieves the content from the origin server that the tenant has defined such HTTP server.

2.1.2 Cloudflare CDN Edge server​:

A CDN edge server is a computer that exists at the logical extreme or “edge” of a network. An edge server often serves as the connection between separate networks. A primary purpose of a CDN edge server is to store content as close as possible to the requesting client machine, thereby reducing latency and improving page load times. An edge server is a type of edge device that provides an entry point into a network. Other edges devices include routers and routing switches. Edge devices are often placed inside Internet exchange points (IxPs) to allow different networks to connect and share transit. There are two types of servers – origin server and the edge server. Origin server gets all the traffic requests directed to it. CDN edge servers store (cache) content in strategic locations in order to take the load off of one or more origin servers. By moving static assets like images, HTML and JavaScript files (and potentially other content) as close as possible to the requesting client machine, an edge

server cache is able to reduce the amount of time it takes for a web resource to load. Origin servers still have an important function to play when using a CDN, as important server-side code such as a database of hashed client credentials used for authentication, typically is maintained at the origin.

As mentioned earlier, normally when a client requests a file from an origin server, the request needs to go roundtrip to that server and back again. A CDN improves the latency by pulling static content files from the origin server into the distributed CDN network in a process called caching. Some CDNs will have advanced features that allow for the selective caching of dynamic content as well. Once the data is cached, the CDN serves the content to the client from the closest CDN data center. The time to download the file from origin server and the edge server is compared by comparing the TTL fields in the static data.

Cloudflare also uses minification (The process of removing the human readable content in the code) of files and file compression. Reliability and redundancy of the service is maintained by load balancing the internet traffic and maintaining the failover check between the primary and the backup servers. In addition to all the functional features the Cloudflare also provides the security by SSL TLS security.

2.1.3 Netflix Open Connect​:

The program’s goal is to provide the content to the customers fast and efficiently by partnering with the ISP’s and maintaining the data centers in the nearby AWS. Every provider should buy an Open Connect Appliance (OCA) and install it near to the ISP location in order to get the better bandwidth and easier maintenance. The OCA’s will be configured with FREE BSD, NGINX and BIRD. NGINX web server is used to get the traffic through HTTP as NGINX (it) is proved for its scalability and performance in delivering content via HTTP. BIRD is a Routing Intelligence Proxy, which is used to share the local ISP network topology to the control system in AWS. The OCA appliances will be having a Flash/HDD storage space and are configured with customized BGP. BGP and Bird work together to know the nearest AWS location to get the content and will fill the content in the off-peak hours. This content is downloaded from either nearby OCA appliances or the next tier NETFLIX server located in nearby AWS. To know the nearby OCA devices the clustering methodology has been used. When two OCA’s are in the same subnet, then they will be considered as same cluster. All this information is analyzed by the BIRD from the BGP messages exchanged between the Netflix’s control system in AWS and the OCA’s. Peer filling and Tier filling are two types of content filling methods used in this process. In Peer filling, the OCA’s in the same subnet can exchange the information without any hassle, but to ensure the peer filling for different subnet OCA’s we need to make sure that the OCA who wants to download the content should be hearing the other OCA subnet via the BGP connection shard with the ISP’s router. The clash between two OCA’s is resolved by Multi Exit Discriminator (MED’s) used in BGP protocol. The Control system in the AWS will monitor the OCA’s health and performance continuously and provides the critical updates required for the devices.

**2.2 VPC:**

Amazon Virtual Private Cloud (Amazon VPC) provides customers with the ability to create as many virtual networks as they need, as well as different options for connecting those networks to each other and to non-AWS infrastructure. One common strategy for connecting multiple, geographically disperse VPCs and remote networks is to create a transit VPC that serves as a global network transit center. A transit VPC simplifies network management and minimizes the number of connections required to connect multiple VPCs and remote networks. remote VPCs access each other and remote networks through the central, transit VPC. The AWS Cloud provides a suite of infrastructure services

that enable you to deploy a transit VPC solution in a highly available, fault-tolerant, and affordable way. By integrating Cisco Cloud Services Router (CSR) with the AWS Cloud, you can take advantage of the functionality of enterprise-class networking services and VPN along with the flexibility and security of AWS. Spoke VPCs are connected to the transit network through dynamically routed VPN connections between their virtual private gateways (VGWs) and the CSR instances. This design uses VPN connections to enable routing between any connected network, including external networks or spoke VPCs in other AWS Regions. VPN connections also allow spoke VPC resources to leverage VGW capabilities for routing and failover in order to maintain highly available network connections to the transit VPC instances. Remote networks also connect to the transit VPC using redundant, dynamically routed VPN connections between their customer gateways and the CSR instances. This design supports dynamic routing protocols, which customers can use to automatically route traffic around potential network failures as well as to propagate network routes to remote networks. Note that all communication with the CSR instances, including the VPN connections between corporate data centers or other provider networks and the transit VPC, uses the transit VPC Internet gateway and the instances’ Elastic IP addresses. Each CSR instance has an associated Amazon CloudWatch alarm that enables automatic recovery of the instance if the underlying EC2 hardware fails. Along with providing direct network routing between VPCs and on-premises networks, this design also enables the transit VPC to implement more complex routing rules, such as network address translation (NAT) between overlapping network ranges, or to add additional network-level packet filtering or inspection. The AWS-to-AWS VPN connectivity in this design relies on software VPN appliances running on the Cisco CSR 1000v instances. It also relies on the capabilities of the hardware VPN device deployed on premises that connects to the software VPN appliance.

**2.3 DNS:**

2.3.1 ROUTE 53

Amazon Route 53 is a highly available and scalable Domain Name System (DNS) web service. Route 53 is an “authoritative DNS” system. An authoritative DNS system provides an update mechanism that developers use to manage their public DNS names. It then answers DNS queries, translating domain names into IP addresses so computers can communicate with each other.

Route 53 features:

1. Register domain names: Route 53 lets you register a name for your website or web application, known as a domain name. After registering the domain name, Route 53 automatically creates a public hosted zone that has the same name as the domain. To route traffic to the resources, records are created, also known as resource record sets, in the hosted zone. Each record includes the following information: name, type, value.
2. Route internet traffic to the resources for your domain: When a user opens a web browser and enters a domain name (linux.com) or subdomain name (apex.linux.com) in the address bar, Route 53 helps connect the browser with your website or web application.
3. Check the health of resources: Route 53 sends automated requests over the internet to a resource, such as a web server, to verify that it's reachable, available, and functional. One can choose to receive notifications when a resource becomes unavailable and choose to route internet traffic away from unhealthy resources. Amazon Route 53 uses following to perform the check: HTTP, HTTPS, or TCP.

1. Management of Route 53 can be done by any means including the web console provided. i.e.; it can be managed via APIs, CLI, SDKs, AWS tools and also third-party tools.

**3. Project Description**​

**3.1 System Description:**

This project will create an infrastructure for tenant by automating a virtual private cloud using open source software and introduces functional and management feature to allow the tenant to host its content on the edge device. This content is accessed by the user by using the system which is mentioned below:

System involves 4 main components from the infrastructure to provide the requested content to the Tenant’s client (user)

1. The DNS Forwarder would be resolving the queries that are destined to the local DNS server of a specific tenant.

It acts as a central DNS server which maintains the information of all the Tenant’s local DNS server.

2. The DNS Server, acts as a Local DNS resolver for a particular Tenant which resolves queries that it receives from the DNS Forwarder.

It would provide the user with the address of the nearest Edge device (i.e. the Edge which is in its zone) to access the content.

3. The Edge Device on its own acts as an edge server, consisting the files that the tenant desires to be put in that locality. The user connects to these devices and asks for a content.

There are three possible scenarios that could be chalked out

Scenario 1: The Edge Device has the content

The device looks up the content based on the request in itself (could be acting as a primary DNS: resolving a hostname to a local filename)

Scenario 2: The Edge device doesn’t have the content

In this case, the Edge device will fetch content from content server using reverse proxy, which in turn provides the content requested by the user.

4. The Content Server hold the manifest files / data of the content that the tenant has to offer.

This would act as the base for all content requirements from the edge devices.

**3.2 Proposed Features:**

**3.2.1 Functional Features:**

**3.2.1.1 Edge service Placement:**

This feature allows tenant to select different subnets in which a particular Edge server is placed. To provide this feature as a service we are taking inputs from tenant as the list of subnet id for corresponding edge server. Thus, this service creates an edge cloud and also gives some redundancy.

**3.2.1.2 Content Delivery API on each Edge server:**

Users connecting to the edge device for acquiring desired content will be provided with the appropriate content according to the needs of the tenant as well as the user. The contents are placed over different edge devices according to the requirements of the tenants. Users will be able to access different content placed on different edge devices. This feature is implemented as a service by allowing tenants to decide whether it wants to introduce this feature at all, and then asking it for Proxy enable which allows content to be fetched from the content server if edge server doesn’t have already.

**3.2.1.3 Selection of Edge service using DNS:**

In this feature, a content query is directed towards an edge server which is intended to be in the same location as that of user. We are using DNS in a hierarchical manner for resolution of content queries. Firstly, when the user requests for a particular content, the query goes to the DNS Forwarder which will resolve query based on tenant name. Secondly, the local DNS server of the tenant on receiving the query from the DNS forwarder will resolve the query based on the user’s source IP address.

**3.2.2 Management Features:**

**3.2.2.1 Accounting Statistics:**

Keeping statistics or logs up-to-date is essential for network growth, from a business point of view. In our case, we are going to concentrate on logging parameters like demand of which edge device and which content in a particular edge device.

There can be a particular edge device or a particular content which is used frequently in a specific area or in a specific edge device respectively. To fulfil this demand, we are going to develop an algorithm which will make the service effective.

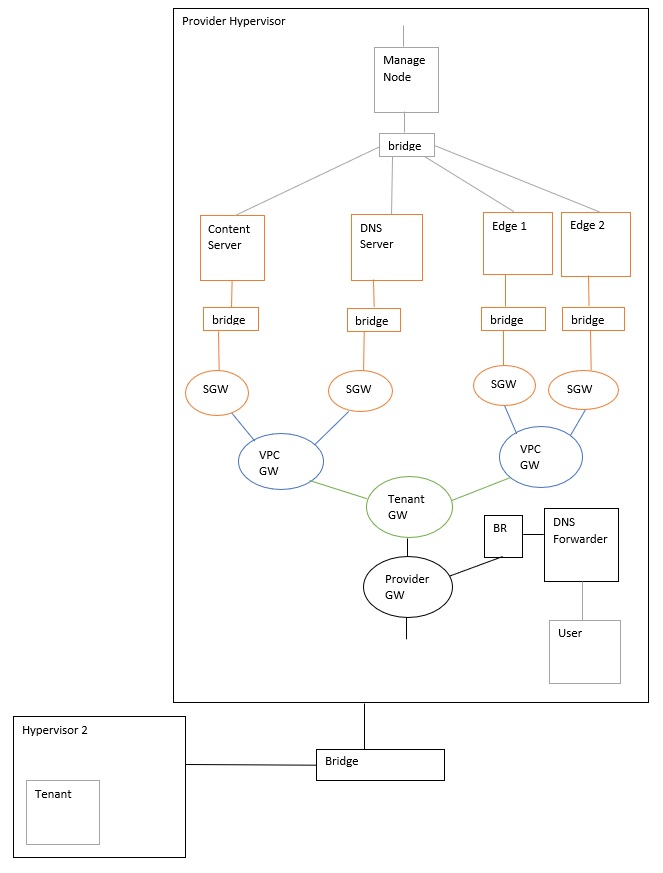
**3.2.1.2 Edge server reliability:**

Reliability is important because the user must be able to send content request to the edge server at all times. For this requirement, the edge server must be active in extreme conditions. But there are some natural limitations for every edge server, which can make the edge device inactive, so to eliminate this issue we are introducing a secondary server at each edge location. Using Keepalived alogrithm, we are going to keep one of the two servers active. If the active edge server goes down than automatically the secondary servers get active. Thus, providing guaranteed edge server reliability.

**3.2.1.3 Edge device self-healing (Kubernetes inspired):**

Availability of edge server at all times is provided by introducing self-healing mechanism in the edge device. If the edge device is exited, it will be started periodically by checking the edge status.

**3.3. Topology Description**



In this project, a cloud service is provided to multiple tenants, such that each tenant can create multiple virtual private cloud and in turn deploy their network inside it. Along with this a DNS and Nginx solution are given, where tenant can access its network and make some desired altercation and also user can request for tenant’s content and acquire it.

in our case “user” can access the content placed on the edge device. Let’s see how does the user access the tenant’s content;

Firstly, when user arrives at the entry point at hypervisor 2 it requests for specific content of the Tenant. A DNS query goes to the DNS forwarder which in turn looks into its DNS records and finds a tenants DNS server for which the request is intended to. Then tenant’s DNS server on receiving the request looks it’s records and gives response which includes nearest Edge’s IP address. Once the request for the content is reached at the Edge, it will resolve by giving the content to the user. Keeping the track of all the content in the Edge and extracting a particular content for each request is done by Nginx web server.

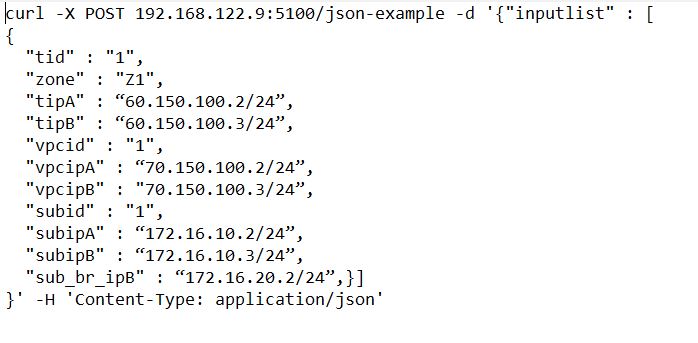
**4. Implementation Architecture**

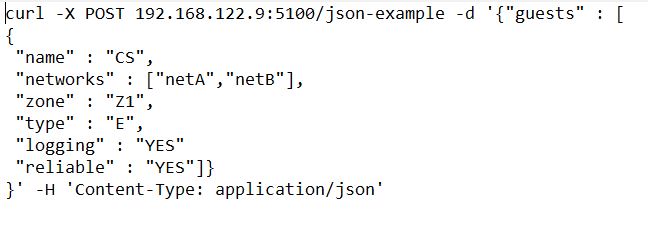
**4.1 Implementation Details**

In our implementation, we have used open source tools and network automation tools like docker Ansible. We have developed a user-defined code to take inputs from the tenant and deployed a virtual private cloud as desired by the tenant. After realizing the difficulties of using VMs, we shifted to docker containers. By using docker we can access some configuration and files, make its image and use it to create container. Also, the trouble of doing ssh every time we access the VM is eliminated in containers as we can do configurations from the hypervisor.

**4.1.1 Northbound:**

In Northbound we are using REST API, which takes input as number of VPCs and number of subnets from the tenant in json format. We allow tenants to build unlimited subnets and VPCs for their infrastructure. By taking IP address of subnets and VPCs from the tenant we assign IP address to namespaces and containers according to their subnets. All the tasks that we are implementing in northbound are same for both VMs and docker containers.





**4.1.2 Southbound:**

In Southbound the process starts with namespace creation for tenant, tenant’s VPC and then subnets. After creating namespaces, containers are created and then connected to the respective subnet namespace. This is followed by automation of DNS server inside one of the containers. The next part is to automate Nginx web server inside all Edge devices. All this automation is done using ansible. During automating the infrastructure using containers we do not need to create network, IP address was easier and no need to create templets and xml file.

**4.1.3 Logical Layer:**

Creation of Infrastructure (VPC, Subnets, containers)

Configure Nginx web server

Configure DNS Server

Tenant container

Tenant Input

In Logical Layer, the process starts by taking inputs from tenants for infrastructure creation using json file. The management node is connected to each and every device in the network so that the control path can be defined through it. By using Tenant container, the tenant will give an input for his requirement. This file is a json file, which contains the information about the number of subnets, number of VPCs and the subnets for each container. The next step is to deploy the infrastructure desired by the Tenants where a script control\_center.py where this python file uses the tenant inputs to create different components of the infrastructure. Control\_center\_final.pyfile calls tenant\_cont.yml file for tenant gateway creation then VPC gateway is created using vpc\_cont.yml file, then subnet gateway creation is done by using subnet\_cont.yml and at the end container is created using end\_cont.yml. After this step, we automated a DNS configuration in two of those created containers and in the final step automation is done to configure Nginx in all the edge devices.

**4.2 Tenant Guide:**

For Tenant, the requested inputs are number of subnets, number of VPCs and subnets for edge device. Thus, allowing tenants to build infrastructure without any limitations.

Along with the above inputs, tenants need to give VPC IP address and Subnet IP address as per their requirement.

**4.3 Developer Guide:**

1. To begin with, the developer must go through section 4.1 thoroughly in order to understand

the entire service.

2. Use json format to take inputs from the Tenant.

3. By using control\_server.py, we are calling all the scripts that are creating different

components of the topology. It is important to make such kind of script for easier

debugging purpose.

4. Use Docker containers to develop the topology and automate it using ansible.

5. It is recommended to use docker container instead of virtual machine because container

provide high performance in compare to VMs

6. If developer wants to add functionality like DNS server or Nginx server. Run scripts like

DNSinstall.sh or nginxinstall.sh and siteconf.sh respectively.

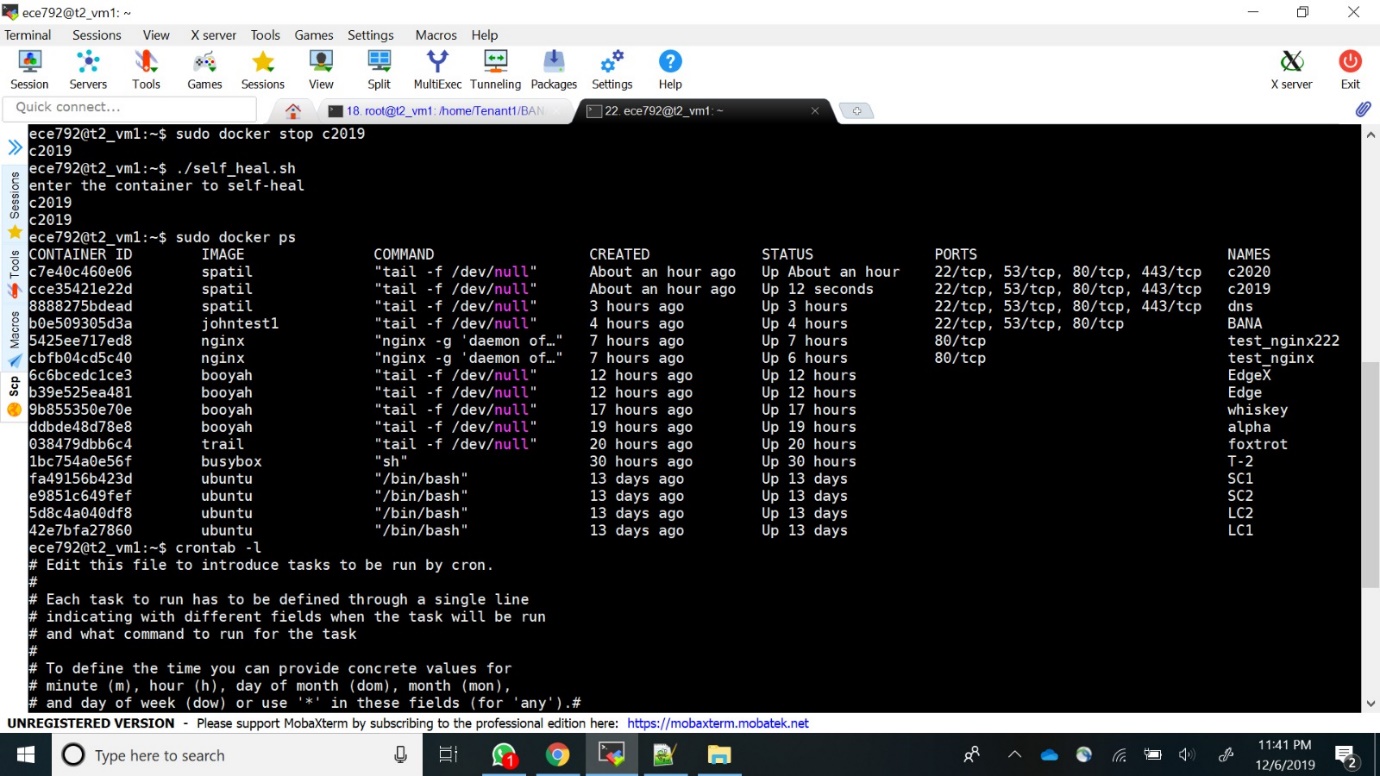
7. If developer wants to add management feature in its system like accounting and reliability.

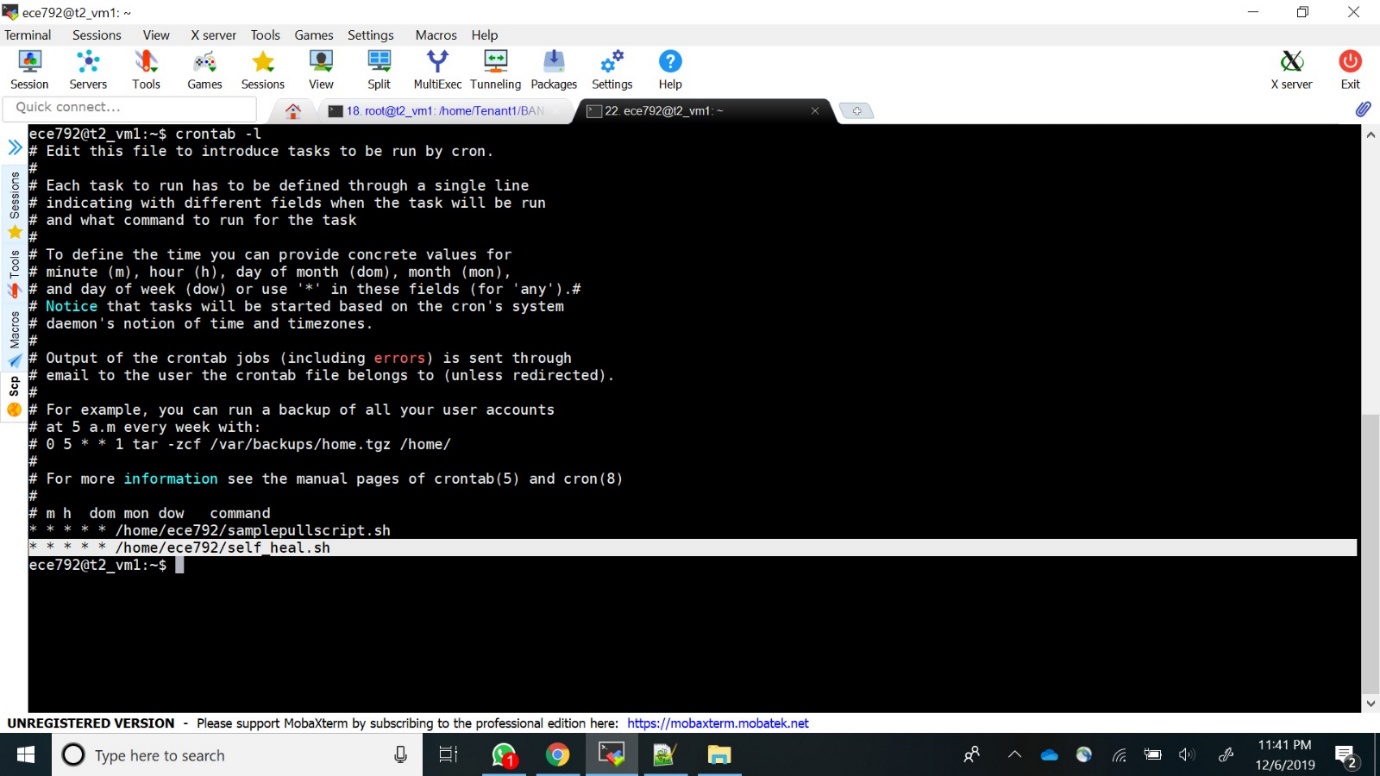
Use scripts like pullaccesslog.sh, addcronjob.sh keep\_new.sh.

**5. Evaluation**

**1. Edge device self-healing**

Edge device goes down when we stop the container. But after a while it gets started automatically. This we can observe in the bellow snap





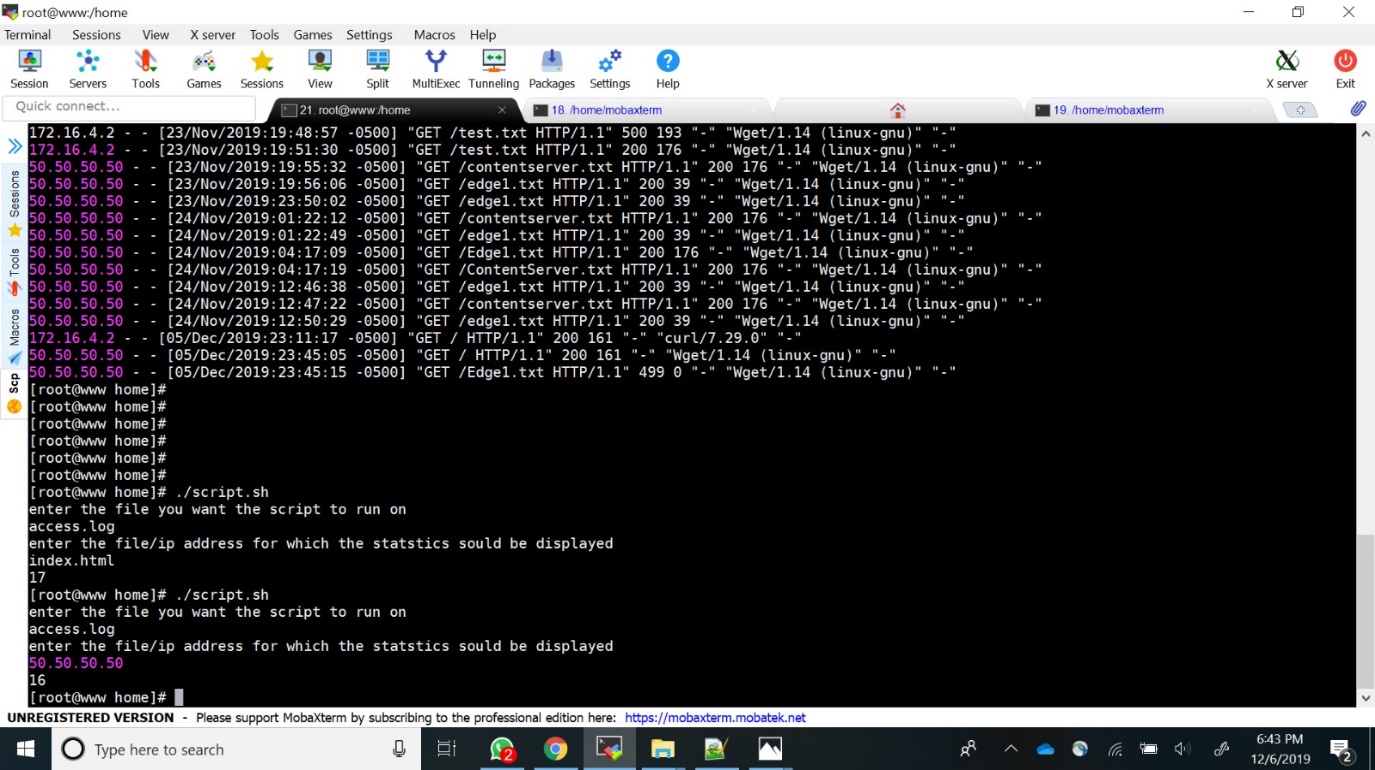
**2. Accounting Statistics**

For accounting, we are pulling all the logs from the Edge servers and the Content Server.

These logs are generated by the nginx server which contains the information of the user’s ip,

Timestamp, File accessed, Error codes. The user can grep the file for specific output. Additionally, we are giving a script for the tenant which asks for the file input and the search content. The search content will be searched in every line in the log file and the count will be returned to the tenant.

This script will be helpful in the automation of finding the no. of requests per movie. The script will also be helpful to get the count of a specific user request based on the IP searched.



**6. Summary**

This project provides CDN service for Edge cloud by automating deployment of virtual private cloud as per Tenant demand and resolve content queries for Tenant’s client (user) with the help of DNS server and Nginx webserver. Tenant demands for an infrastructure by giving some inputs through a json file. Using this file, our service automates Tenant’s network configuration and creates a virtual private cloud for the Tenant. By building a DNS server and Nginx webserver in this cloud, we allow user to request for a content and successfully receive the same. DNS server is implemented in a hierarchical and recursive way to resolve user’s query and Nginx webserver fetches the content from the file system inside the Edge device. Our service will be active all the time even if the Edge device goes down, the solution for this condition is to redirect the queries to the secondary Edge device. Along with fault management, we are going to carry out some analysis about Edge device usage and specific content usage which will give the Tenant a picture of its network and content utilization. This feature will allow Tenant to make necessary changes wherever needed. To keep the tenant data isolated from the outer network we have introduced tunneling in the network.

**7. Future Scope**

1. To provide CDN service we have used Edge devices as a server to store the content, so if

the Edge does not have a particular content, the request will be redirected to the content

serve. So, we can make this solution better by redirecting the request to the nearest Edge

which has the requested content, thus increasing the performance.

2. Dynamic edge placement can be implemented by monitoring the logs and checking for

saturated edge server.

3. To make the analytics more presentable for the Tenant we can use open source

virtualization tools like Grafana.